Networking Properties

There are a few standard system properties used to alter the mechanisms and behavior of the various classes of the java.net package. Some are checked only once at startup of the VM, and therefore are best set using the -D option of the java command, while others have a more dynamic nature and can also be changed using the System.setProperty() API. The purpose of this document is to list and detail all of these properties.

If there is no special note, a property value is checked every time it is used.

IPv4 / IPv6

- **java.net.preferIPv4Stack** (default: false)
  If IPv6 is available on the operating system the underlying native socket will be, by default, an IPv6 socket which lets applications connect to, and accept connections from, both IPv4 and IPv6 hosts. However, in the case an application would rather use IPv4 only sockets, then this property can be set to **true**. The implication is that it will not be possible for the application to communicate with IPv6 only hosts.

- **java.net.preferIPv6Addresses** (default: false)
  When dealing with a host which has both IPv4 and IPv6 addresses, and if IPv6 is available on the operating system, the default behavior is to prefer using IPv4 addresses over IPv6 ones. This is to ensure backward compatibility, for example applications that depend on the representation of an IPv4 address (e.g. 192.168.1.1). This property can be set to **true** to change that preference and use IPv6 addresses over IPv4 ones where possible.

Both of these properties are checked only once, at startup.

Proxies

A proxy server allows indirect connection to network services and is used mainly for security (to get through firewalls) and performance reasons (proxies often do provide caching mechanisms). The following properties allow for configuration of the various type of proxies.

- HTTP

  The following proxy settings are used by the HTTP protocol handler

- **http.proxyHost** (default: <none>)
  The hostname, or address, of the proxy server

- **http.proxyPort** (default: 80)
  The port number of the proxy server.

- **http.nonProxyHosts** (default: localhost|127.*|[::1])
  Indicates the hosts that should be accessed without going through the proxy. Typically this defines internal hosts. The value of this property is a list of hosts, separated by the '|' character. In addition the wildcard character '*' can be used for pattern matching. For example `-Dhttp.nonProxyHosts=”*.foo.com|localhost”` will indicate that every hosts in the foo.com domain and the localhost should be accessed directly even if a proxy server is specified.

  The default value excludes all common variations of the loopback address.

- HTTPS
  This is HTTP over SSL, a secure version of HTTP mainly used when confidentiality (like on payment sites) is needed.

  The following proxy settings are used by the HTTPS protocol handler.

  - **https.proxyHost**(default: <none>)
    The hostname, or address, of the proxy server

  - **https.proxyPort** (default: 443)
    The port number of the proxy server.

    The HTTPS protocol handler will use the same nonProxyHosts property as the HTTP protocol.

- FTP

  The following proxy settings are used by the FTP protocol handler.

  - **ftp.proxyHost**(default: <none>)
    The hostname, or address, of the proxy server

  - **ftp.proxyPort** (default: 80)
    The port number of the proxy server.

  - **ftp.nonProxyHosts** (default: localhost|127.*|[::1])

Indicates the hosts that should be accessed without going through the proxy. Typically this defines internal hosts. The value of this property is a list of hosts, separated by the '|' character. In addition the wildcard character '*' can be used for pattern matching. For example `-Dhttp.nonProxyHosts=”*.foo.com|localhost”` will indicate that every hosts in the foo.com domain and the localhost should be accessed directly even if a proxy server is specified.

The default value excludes all common variations of the loopback address.

- SOCKS
This is another type of proxy. It allows for lower level type of tunneling since it works at the TCP level. In effect, in the Java(tm) platform setting a SOCKS proxy server will result in all TCP connections to go through that proxy, unless other proxies are specified. If SOCKS is supported by a Java SE implementation, the following properties will be used:

    - **socksProxyHost** (default: <none>)
    The hostname, or address, of the proxy server.

    - **socksProxyPort** (default: 1080)
    The port number of the proxy server.

    - **socksProxyVersion** (default: 5)
    The version of the SOCKS protocol supported by the server. The default is `5` indicating SOCKS V5, alternatively `4` can be specified for SOCKS V4. Setting the property to values other than these leads to unspecified behavior.

    - **java.net.socks.username** (default: <none>)
    Username to use if the SOCKSv5 server asks for authentication and no java.net.Authenticator instance was found.

    - **java.net.socks.password** (default: <none>)
    Password to use if the SOCKSv5 server asks for authentication and no java.net.Authenticator instance was found.

      Note that if no authentication is provided with either the above properties or an Authenticator, and the proxy requires one, then the **user.name** property will be used with no password.

- **java.net.useSystemProxies** (default: false)
On recent Windows systems and on Gnome 2.x systems it is possible to tell the java.net stack, setting this property to **true**, to use the system proxy settings (both these systems let you set proxies globally through their user interface). Note that this property is checked only once at startup.

Misc HTTP properties

- **http.agent** (default: “Java/<version>”)
  Defines the string sent in the User-Agent request header in http requests. Note that the string “Java/<version>” will be appended to the one provided in the property (e.g. if -Dhttp.agent=”foobar” is used, the User-Agent header will contain “foobar Java/1.5.0” if the version of the VM is 1.5.0). This property is checked only once at startup.

- **http.keepalive** (default: true)
  Indicates if persistent connections should be supported. They improve performance by allowing the underlying socket connection to be reused for multiple http requests. If this is set to true then persistent connections will be requested with HTTP 1.1 servers.

- **http.maxConnections** (default: 5)
  If HTTP keepalive is enabled (see above) this value determines the maximum number of idle connections that will be simultaneously kept alive, per destination.

- **http.keepAlive.time.server** and **http.keepAlive.time.proxy**

  These properties modify the behavior of the HTTP keepalive cache in the case where the server (or proxy) has not specified a keepalive time. If the property is set in this case, then idle connections will be closed after the specified number of seconds. If the property is set, and the server does specify a keepalive time in a "Keep-Alive" response header, then the time specified by the server is used. If the property is not set and also the server does not specify a keepalive time, then connections are kept alive for an implementation defined time, assuming {@code http.keepAlive} is {@code true}.

- **http.maxRedirects** (default: 20)
  This integer value determines the maximum number, for a given request, of HTTP redirects that will be automatically followed by the protocol handler.

- **http.auth.digest.validateServer** (default: false)

- **http.auth.digest.validateProxy** (default: false)

- **http.auth.digest.cnonceRepeat** (default: 5)

  These 3 properties modify the behavior of the HTTP digest authentication mechanism. Digest authentication provides a limited ability for the server to authenticate itself to the client (i.e. By proving it knows the user's password). However not all HTTP servers support this capability and by default it is turned off. The first two properties can be set to true to enforce this check for authentication with either an origin or proxy server, respectively.

It is usually not necessary to change the third property. It determines how many times a cnonce value is re-used. This can be useful when the MD5-sess algorithm is being used. Increasing this value reduces the computational overhead on both client and server by reducing the amount of material that has to be hashed for each HTTP request.

- **http.auth.ntlm.domain** (default: <none>)
  NTLM is another authentication scheme. It uses the java.net.Authenticator class to acquire usernames and passwords when they are needed. However NTLM also needs the NT domain name. There are 3 options for specifying that domain:

  1. Do not specify it. In some environments the domain is actually not required and the application does not have to specify it.

  2. The domain name can be encoded within the username by prefixing the domain name, followed by a back-slash '\' before the username. With this method existing applications that use the authenticator class do not need to be modified, as long as users are made aware that this notation must be used.

  3. If a domain name is not specified as in method 2) and these property is defined, then its value will be used a the domain name.

- **jdk.https.negotiate.cbt** (default: <never>)
  This controls the generation and sending of TLS channel binding tokens (CBT) when Kerberos or the Negotiate authentication scheme using Kerberos are employed over HTTPS with {@code HttpsURLConnection}. There are three possible settings:

  1. "never". This is also the default value if the property is not set. In this case, CBTs are never sent.

  2. "always". CBTs are sent for all Kerberos authentication attempts over HTTPS.

  3. "domain:<comma separated domain list>" Each domain in the list specifies destination host or hosts for which a CBT is sent. Domains can be single hosts like foo, or foo.com, or literal IP addresses as specified in RFC 2732, or wildcards like *.foo.com which matches all hosts under foo.com and its sub-domains. CBTs are not sent to any destinations that don't match one of the list entries

  The channel binding tokens generated are of the type "tls-server-end-point" as defined in RFC 5929.

- **jdk.http.maxHeaderSize** (default: 393216 or 384kB)
  This is the maximum header field section size that a client is prepared to accept. This is computed as the sum of the

size of the uncompressed header name, plus the size of the uncompressed header value, plus an overhead of 32 bytes for each field section line. If a peer sends a field section that exceeds this size a {@link java.net.ProtocolException ProtocolException} will be raised. This applies to all versions of the HTTP protocol. A value of zero or a negative value means no limit. If left unspecified, the default value is 393216 bytes.

All these properties are checked only once at startup.

Address Cache

The java.net package, when doing name resolution, uses an address cache for both security and performance reasons. Any address resolution attempt, be it forward (name to IP address) or reverse (IP address to name), will have its result cached, whether it was successful or not, so that subsequent identical requests will not have to access the naming service. These properties allow for some tuning on how the cache is operating.

- **networkaddress.cache.ttl** (default: see below)
  Value is an integer corresponding to the number of seconds successful name lookups will be kept in the cache. A value of -1, or any other negative value for that matter, indicates a “cache forever” policy, while a value of 0 (zero) means no caching. The default value is -1 (forever) if a security manager is installed, and implementation specific when no security manager is installed.

- **networkaddress.cache.negative.ttl** (default: 10)
  Value is an integer corresponding to the number of seconds an unsuccessful name lookup will be kept in the cache. A value of -1, or any negative value, means “cache forever”, while a value of 0 (zero) means no caching.

Since these 2 properties are part of the security policy, they are not set by either the -D option or the System.setProperty() API, instead they are set as security properties.